US009459923B2

(12) **United States Patent**
Branson et al.

(10) **Patent No.: US 9,459,923 B2**
(45) **Date of Patent: *Oct. 4, 2016**

(54) **DYNAMIC RUN TIME ALLOCATION OF DISTRIBUTED JOBS WITH APPLICATION SPECIFIC METRICS**

(71) Applicant: **International Business Machines Corporation**, Armonk, NY (US)

(72) Inventors: **Michael J. Branson**, Rochester, MN (US); **John M. Santosuosso**, Rochester, MN (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 646 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/900,948**

(22) Filed: **May 23, 2013**

(65) **Prior Publication Data**

US 2013/0254777 A1 Sep. 26, 2013

Related U.S. Application Data

(63) Continuation of application No. 12/837,568, filed on Jul. 16, 2010, now Pat. No. 8,566,837.

(51) **Int. Cl.**
G06F 9/46 (2006.01)
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 9/505*** (2013.01); ***G06F 9/50*** (2013.01); ***G06F 9/5066*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,386,586	B1 *	6/2008	Headley	G06F 9/5033 709/201
8,566,837	B2 *	10/2013	Branson	G06F 9/50 709/208
2006/0167984	A1	7/2006	Fellenstein et al.	
2008/0104605	A1	5/2008	Steinder et al.	
2008/0216087	A1	9/2008	Dillenberger et al.	
2009/0048998	A1	2/2009	Kaminsky et al.	

(Continued)

OTHER PUBLICATIONS

U.S. Appl. No. 13/900,948 entitled "Dynamic Run Time Allocation of Distributed Jobs With Application Specific Metrics" filed May 23, 2013 by Michael J. Branson et al.

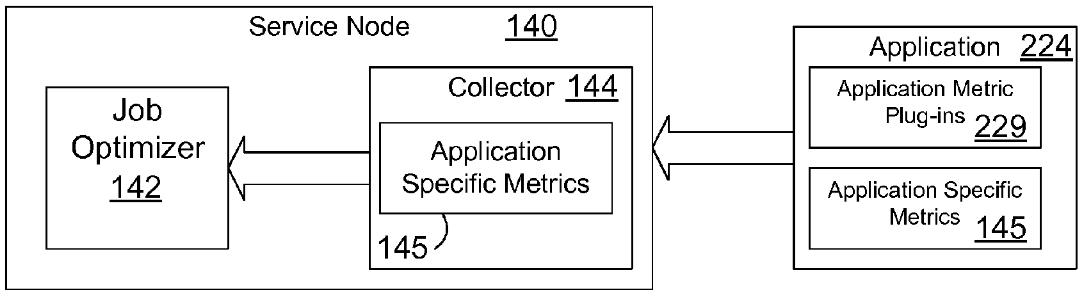
(Continued)

Primary Examiner — Kenneth Tang
(74) *Attorney, Agent, or Firm* — Martin & Associates, LLC; Bret J. Petersen

(57) **ABSTRACT**

A job optimizer dynamically changes the allocation of processing units on a multi-nodal computer system. A distributed application is organized as a set of connected processing units. The arrangement of the processing units is dynamically changed at run time to optimize system resources and interprocess communication. A collector collects application specific metrics determined by application plug-ins. A job optimizer analyzes the collected metrics and determines how to dynamically arrange the processing units within the jobs. The job optimizer may determine to combine multiple processing units into a job on a single node when there is an overutilization of an interprocess communication between processing units. Alternatively, the job optimizer may determine to split a job's processing units into multiple jobs on different nodes where one or more of the processing units are over utilizing the resources on the node.

11 Claims, 6 Drawing Sheets



Service Node 140
Job Optimizer 142
Collector 144
Application Specific Metrics
145
Application 224
Application Metric Plug-ins 229
Application Specific Metrics 145

(56) **References Cited**

U.S. PATENT DOCUMENTS

2009/0083390	A1	3/2009	Abu-Ghazaleh et al.	
2009/0150898	A1	6/2009	Sohn et al.	
2009/0158276	A1	6/2009	Barsness et al.	
2009/0313636	A1	12/2009	Barsness et al.	
2010/0011254	A1	1/2010	Votta et al.	
2010/0043009	A1*	2/2010	Marchand	G06F 9/4881 718/104
2010/0125477	A1	5/2010	Mousseau et al.	
2010/0131959	A1*	5/2010	Spiers	G06F 9/50 718/105

OTHER PUBLICATIONS

U.S. Appl. No. 12/821,784 entitled "Dynamic Run Time Allocation of Distributed Jobs" filed Jun. 23, 2010 by Michael J. Branson et al.

U.S. Appl. No. 13/755,146 entitled "Dynamic Run Time Allocation of Distributed Jobs" filed Jan. 31, 2013 by Michael J. Branson et al.

U.S. Appl. No. 13/709,222 entitled "Dynamic Run Time Allocation of Distributed Jobs With Application Specific Metrics" filed Dec. 10, 2012 by Michael J. Branson et al.

* cited by examiner

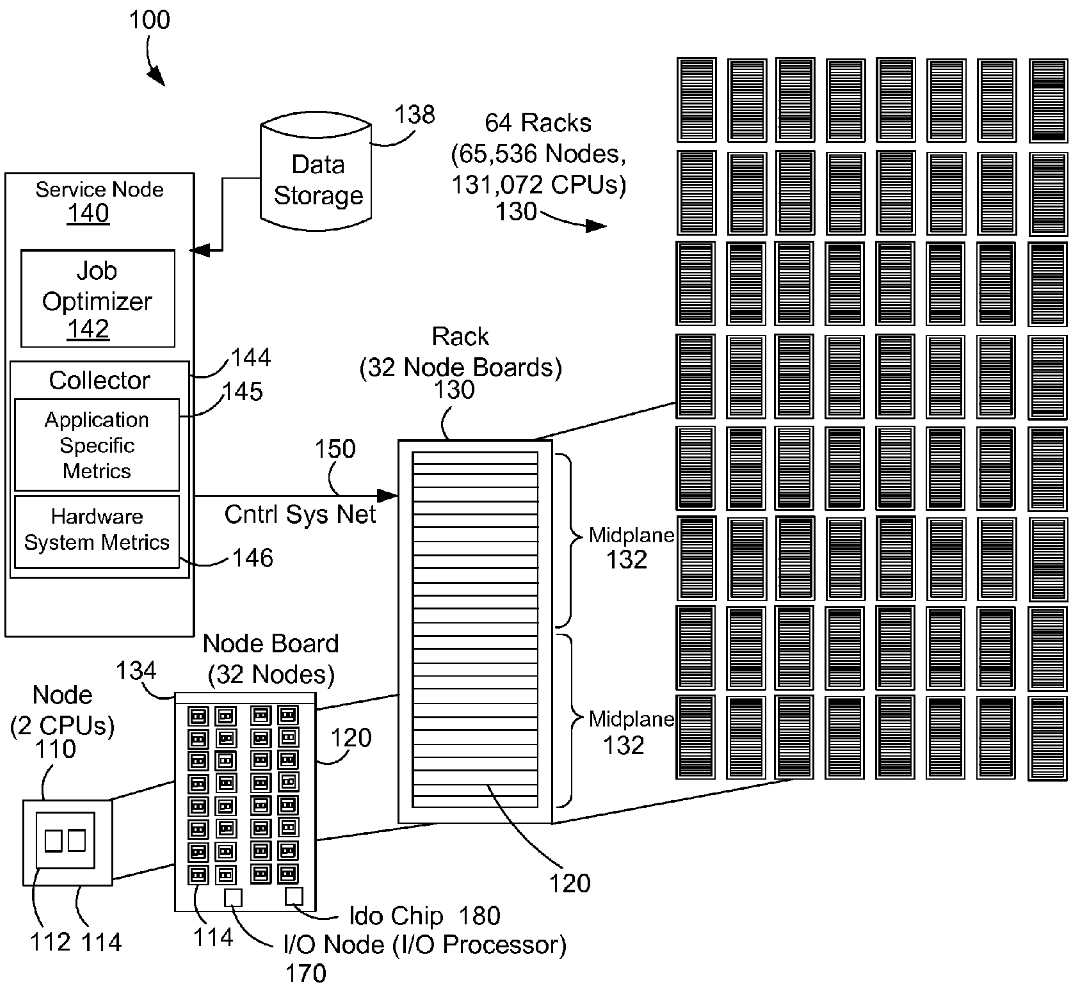

100
138
Data Storage
64 Racks (65,536 Nodes, 131,072 CPUs)
130
Service Node
140
Job Optimizer
142
Collector
144
145
Application Specific Metrics
Hardware System Metrics
146
Rack (32 Node Boards)
130
150
Cntrl Sys Net
Midplane
132
Midplane
132
Node Board (32 Nodes)
134
Node (2 CPUs)
110
120
120
112 114
114
Ido Chip 180
I/O Node (I/O Processor)
170

FIG. 1

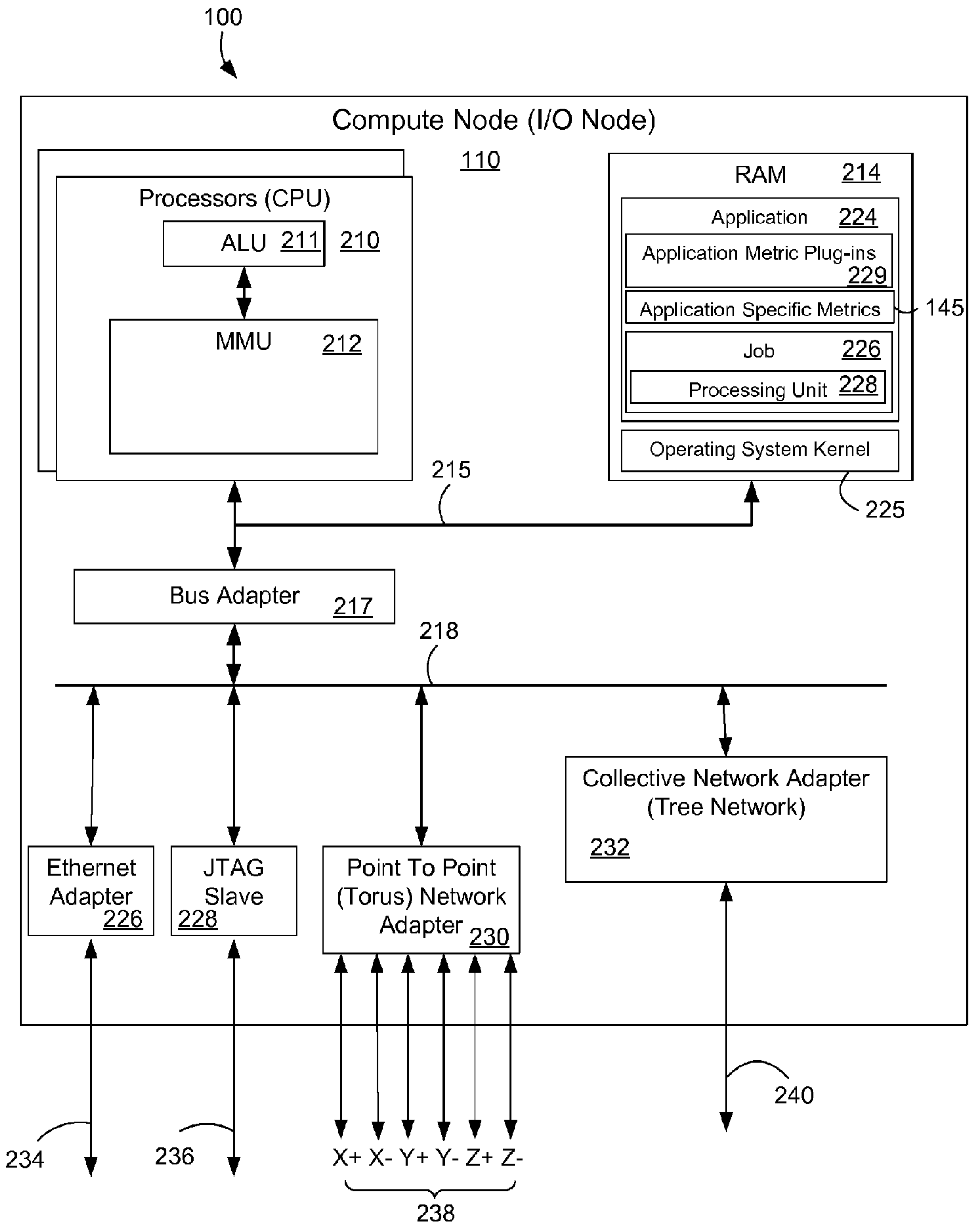

100
Compute Node (I/O Node)
110
Processors (CPU)
ALU 211
210
MMU 212
RAM 214
Application 224
Application Metric Plug-ins 229
Application Specific Metrics
145
Job 226
Processing Unit 228
Operating System Kernel
225
215
Bus Adapter 217
218
Ethernet Adapter 226
JTAG Slave 228
Point To Point (Torus) Network Adapter 230
Collective Network Adapter (Tree Network) 232
234
236
X+ X- Y+ Y- Z+ Z-
238
240

FIG. 2

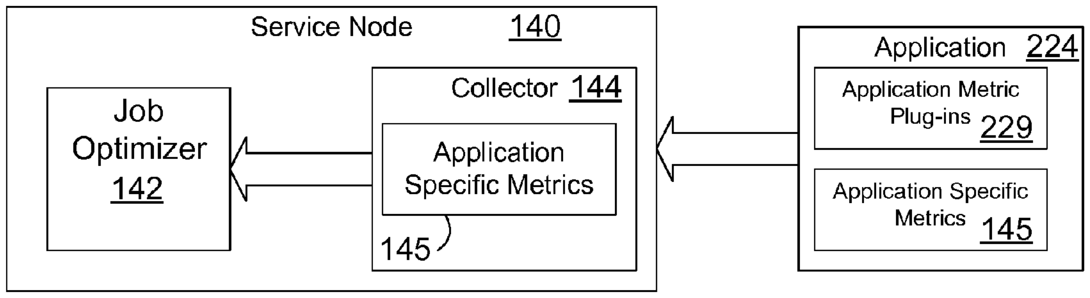

Service Node 140
Job Optimizer 142
Collector 144
Application Specific Metrics
145
Application 224
Application Metric Plug-ins 229
Application Specific Metrics 145

FIG. 3

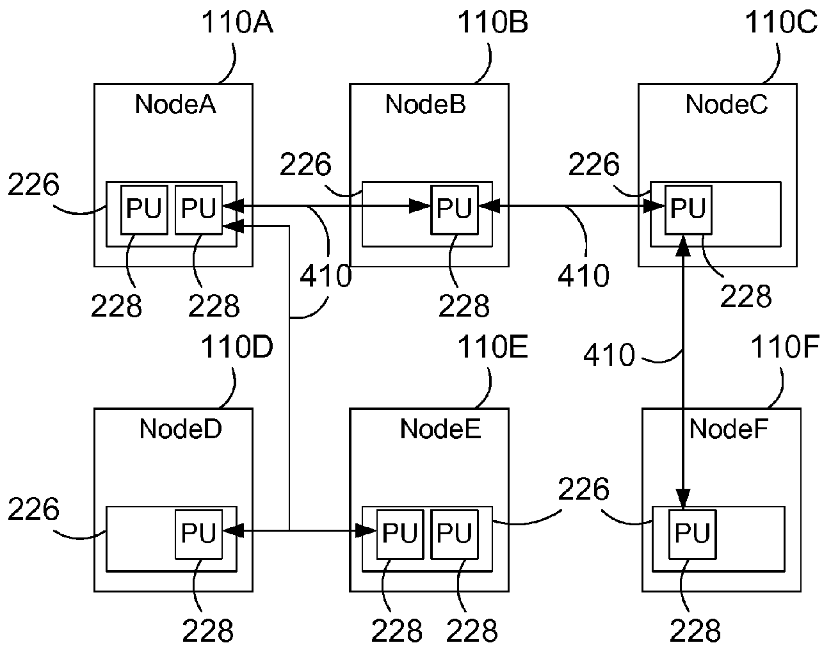

110A
110B
110C
NodeA
NodeB
NodeC
226
226
226
PU
PU
PU
PU
228
228
410
410
228
228
410
110D
110E
110F
NodeD
NodeE
NodeF
226
226
PU
PU
PU
PU
228
228
228
228

FIG. 4

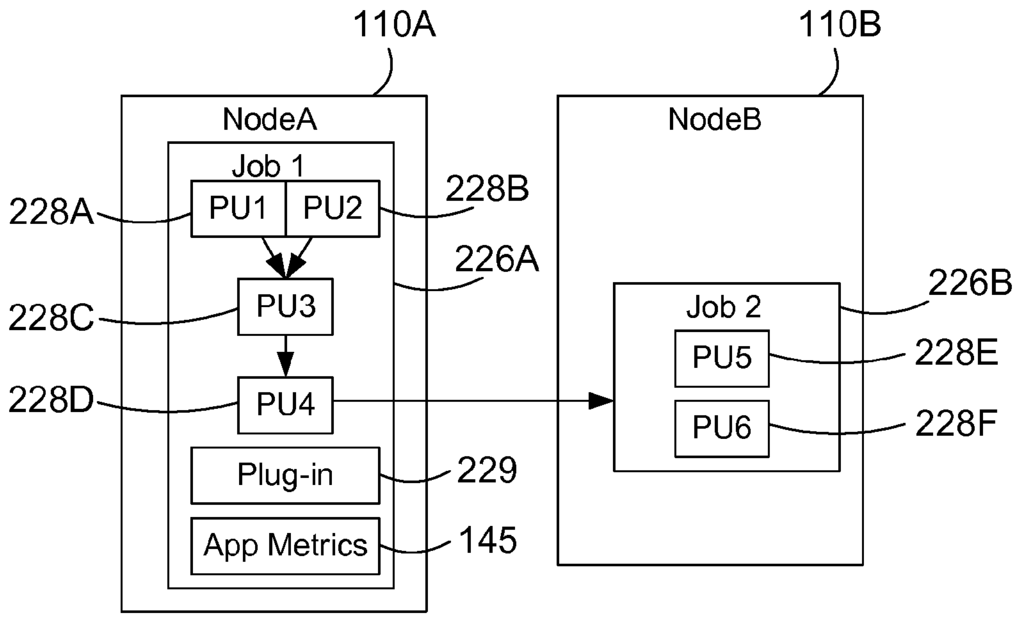

110A
110B
NodeA
Job 1
PU1
PU2
PU3
PU4
Plug-in
App Metrics
NodeB
Job 2
PU5
PU6
228A
228B
226A
228C
228D
229
145
226B
228E
228F

FIG. 5

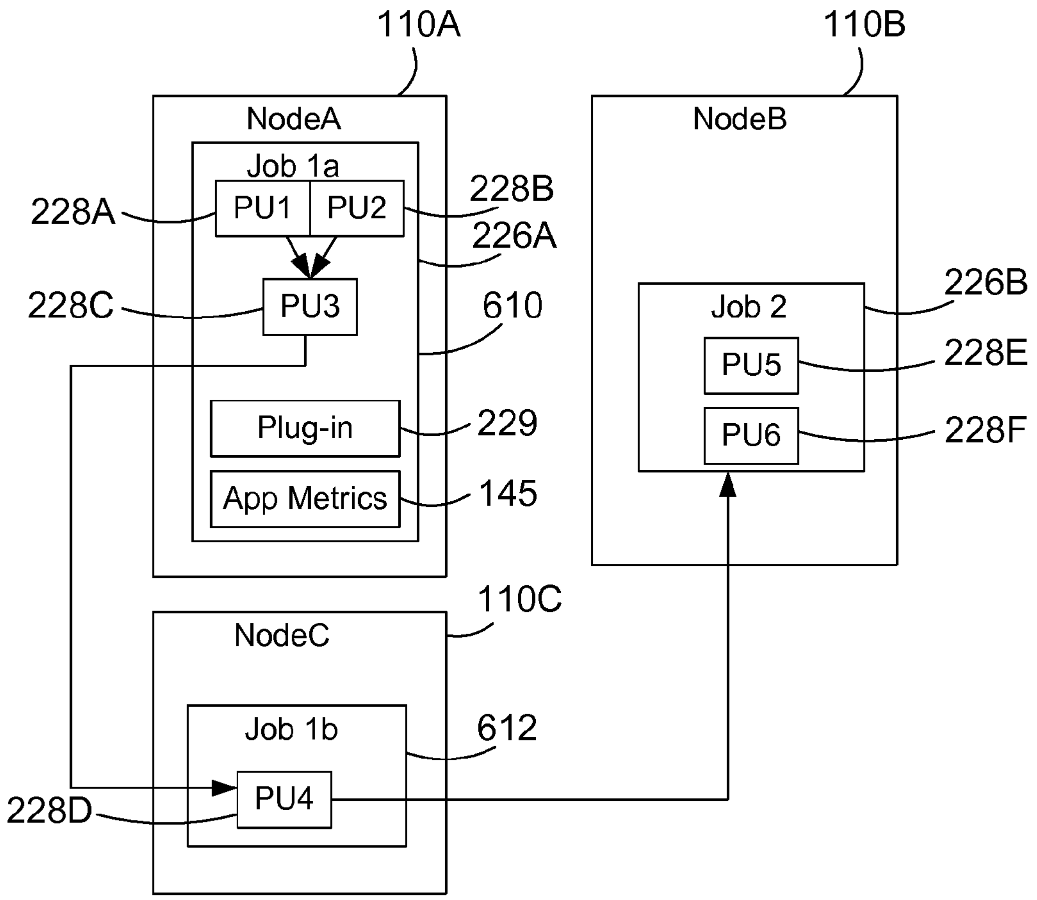

110A
110B
NodeA
Job 1a
PU1
PU2
PU3
Plug-in
App Metrics
NodeB
Job 2
PU5
PU6
NodeC
Job 1b
PU4
110C
228A
228B
226A
228C
610
229
145
226B
228E
228F
612
228D

FIG. 6

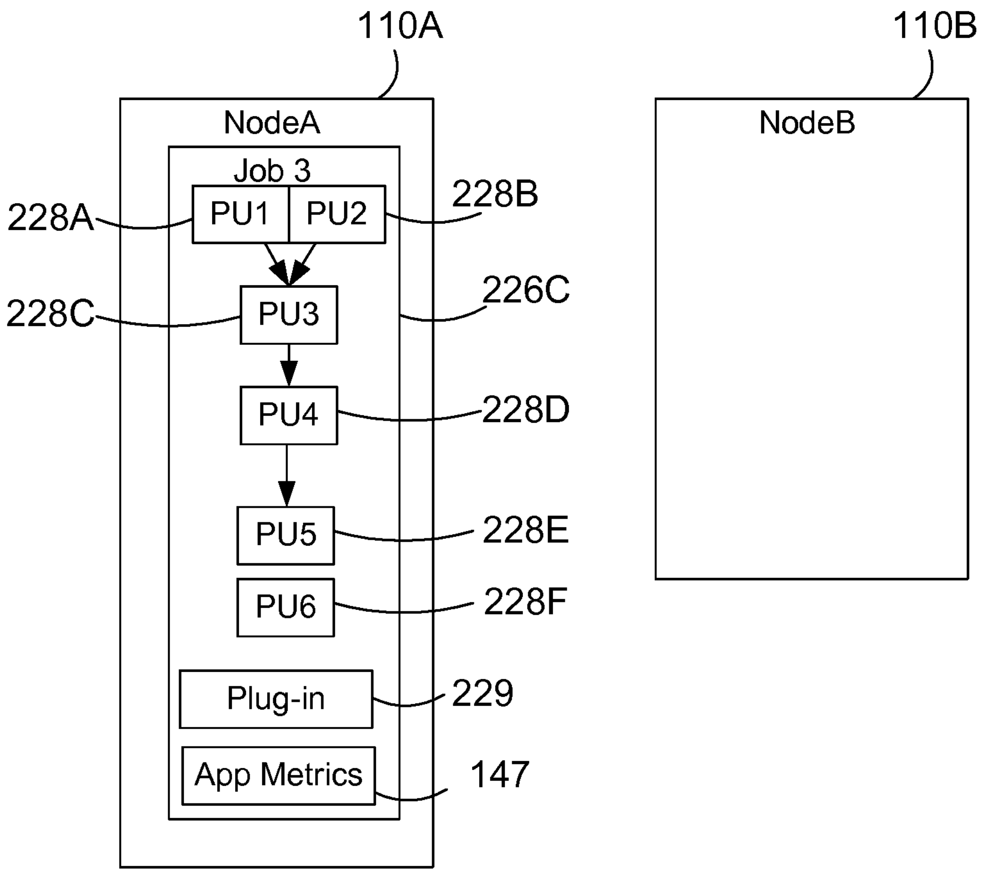

110A
110B
NodeA
NodeB
Job 3
PU1
PU2
PU3
PU4
PU5
PU6
Plug-in
App Metrics
228A
228B
228C
226C
228D
228E
228F
229
147

FIG. 7

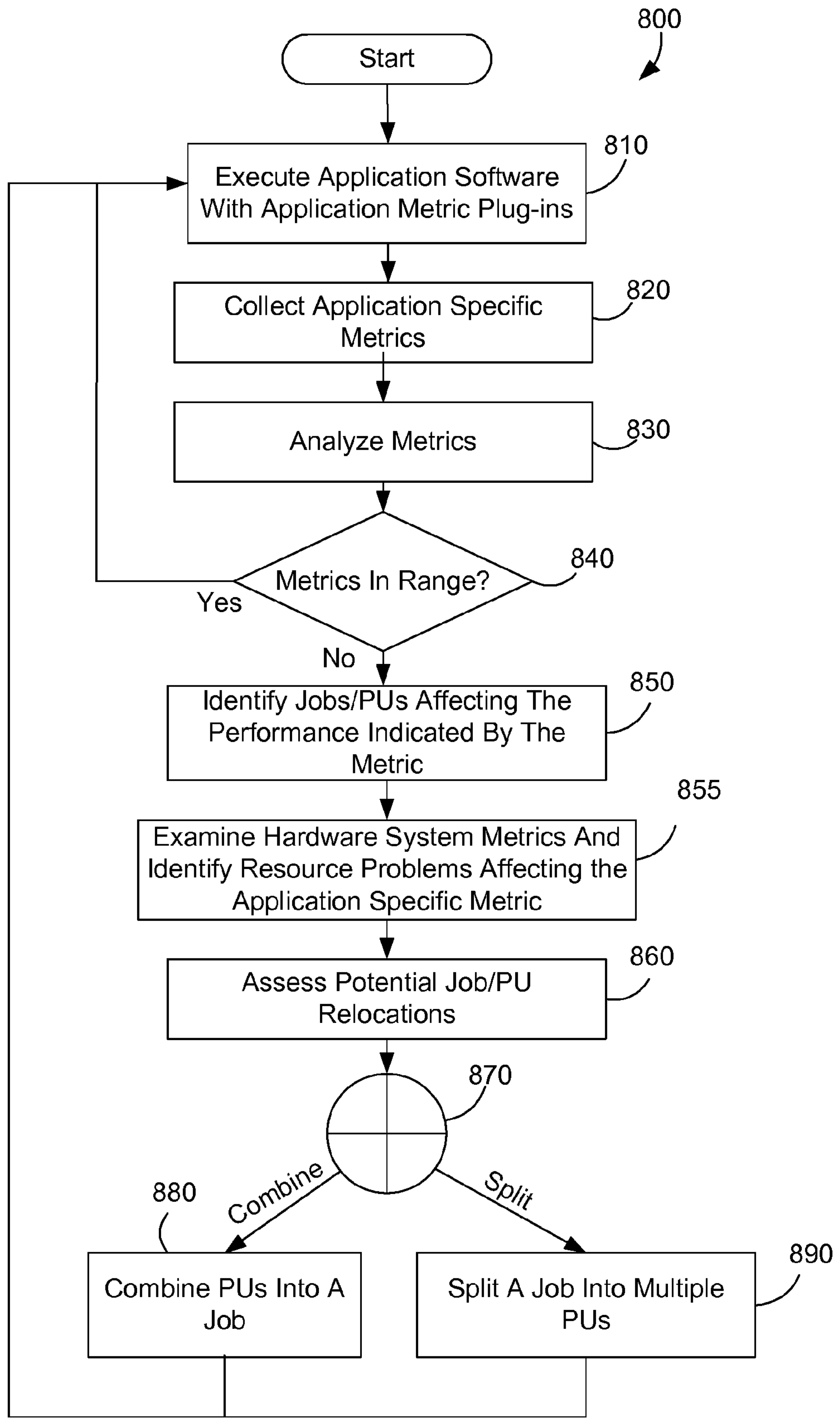

800
Start
Execute Application Software With Application Metric Plug-ins
810
Collect Application Specific Metrics
820
Analyze Metrics
830
Metrics In Range?
840
Yes
No
Identify Jobs/PUs Affecting The Performance Indicated By The Metric
850
Examine Hardware System Metrics And Identify Resource Problems Affecting the Application Specific Metric
855
Assess Potential Job/PU Relocations
860
870
Combine
Split
880
Combine PUs Into A Job
Split A Job Into Multiple PUs
890

FIG. 8

DYNAMIC RUN TIME ALLOCATION OF DISTRIBUTED JOBS WITH APPLICATION SPECIFIC METRICS

BACKGROUND OF THE INVENTION

1. Technical Field

This disclosure generally relates to parallel computing systems, and more specifically relates to dynamically allocating a job or a processing unit (part of a job) on a multi-nodal, parallel computer system based on application specific metrics.

2. Background Art

Large, multi-nodal computer systems (e.g. grids, supercomputers, commercial clusters, etc.) continue to be developed to tackle sophisticated computing jobs. One such multi-nodal parallel computer being developed by International Business Machines Corporation (IBM) is the Blue Gene system. The Blue Gene system is a scalable system with 65,536 or more compute nodes. Each node consists of a single ASIC (application specific integrated circuit) and memory. Each node typically has 512 megabytes of local memory. The full computer is housed in 64 racks or cabinets with 32 node boards in each. Each node board has 32 processors and the associated memory for each processor. As used herein, a massively parallel computer system is a system with more than about 10,000 processor nodes.

These new systems are dramatically changing the way programs and businesses are run. Because of the large amounts of data needing to be processed, current systems simply cannot keep up with the workload. The computer industry is more and more using distributed capacity or distributed computing. An application or sometimes a part of an application is often referred to as a "job". In distributed computing, a job may be broken up into separate run time units (referred to herein as processing units) and executed on different nodes of the system. The processing units are assigned to a node in the distributed system by a job scheduler or job optimizer.

DISCLOSURE OF INVENTION

A method and apparatus is described for a job optimizer that dynamically changes the distribution of processing units on a multi-nodal computer system using application specific metrics. A distributed application is organized as a set of connected processing units. The arrangement of the processing units is dynamically changed at run time to optimize system resources and interprocess communication. A collector collects application specific metrics determined by application plug-ins. A job optimizer analyzes the collected application specific metrics, and then examines needed hardware specific metrics to determine how to dynamically arrange the processing units within the jobs. The job optimizer may determine to combine multiple processing units into a job on a single node when there is an overutilization of an interprocess communication between processing units. Alternatively, the job optimizer may determine to split a job's processing units into multiple jobs on different nodes where one or more of the processing units are over utilizing the resources on the node.

The disclosed embodiments are directed to the Blue Gene architecture but can be implemented on any cluster with a high speed interconnect that can perform broadcast communication. The foregoing and other features and advantages

will be apparent from the following more particular description, as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF DRAWINGS

The disclosure will hereinafter be described in conjunction with the appended drawings, where like designations denote like elements, and:

FIG. **1** is a block diagram of a computer system as described herein;

FIG. **2** is a block diagram of a single node of a massively parallel computer system as described herein;

FIG. **3** is a block diagram that illustrates the interaction of the software elements described herein;

FIG. **4** is a block diagram representing a portion of the computer system **100** shown in FIG. **1**;

FIG. **5** is a block diagram representing two nodes of a computer system as represented in FIG. **1** to illustrate an example of dynamically allocating an application or job as described herein;

FIG. **6** is a block diagram similar to FIG. **5** to illustrate an example of dynamically dividing an application or job as described herein;

FIG. **7** is a block diagram similar to FIG. **5** to illustrate an example of dynamically combining an application or job as described herein; and

FIG. **8** is a method flow diagram for dynamically allocating an application or job as described herein.

BEST MODE FOR CARRYING OUT THE INVENTION

In this disclosure, a method and apparatus is described for a job optimizer that dynamically changes the allocation of processing units (PU) on a multi-nodal computer system based on application specific metrics. A distributed application is organized as a set of connected processing units. The arrangement of the processing units is dynamically changed at run time to optimize system resources and interprocess communication. A collector collects application specific metrics determined by application plug-ins. A job optimizer analyzes the collected application specific metrics, and then examines needed hardware specific metrics to determine how to dynamically arrange the processing units within the jobs. The job optimizer may determine to combine multiple processing units into a job on a single node when there is an overutilization of an interprocess communication between processing units. Alternatively, the job optimizer may determine to split a job's processing units into multiple jobs on different nodes where one or more of the processing units are over utilizing the resources on the node. In addition, the job optimizer may determine to split a job's processing units into multiple jobs on the same node, in order to better utilize a node with multiple processors.

In a distributed environment, message passing and shared memory become standard mechanisms to address how information is passed back and forth between processes or processing units. When writing distributed applications, developers typically need to design up-front how information is passed between its distributed parts. Likewise, some distributed systems are set up such that at deploy time the end user can pick how processes will communicate. When applications are distributed in a multi nodal environment, trade-offs are typically made to determine where segments of the application, i.e. "jobs" should be broken into separate processing units (sometimes referred to as run time units) or kept together in one job such that they can communicate more efficiently with each other. One drawback of processing units in separate jobs, is that it increases IPC (Interprocess communications) on mechanisms for communication such as shared memory or information protocol sockets (IP) which has a negative impact on performance. Alternatively, where these processing units are kept together in one job they can use some sort of synchronization method of passing or accessing data amongst a plurality of threads. While the environment dictates the optimal tradeoff between these, it's really not possible to know exactly what the environment will be like at run time as things change and evolve as data gets processed. The job optimizer as described herein can dynamically reorganize the allocation processing units based on a changing environment as discovered by collecting application specific metrics determined by the application.

The dynamic allocation of processing units as described herein is facilitated by a software system that provides an environment for distributed computing with local/remote transparency to the application developer. This "software system" could be part of an operating system, or it could be a layer of software running on an operating system. The software system typically will utilize more efficient communication mechanisms in the local case than in the remote cases. The application code is written in a manner that is indifferent as to whether a PU is communicating (i.e. exchanging data) with another PU via an intra-process mechanism (i.e. stack, heap, etc.), an inter-process mechanism (e.g. TCP/IP socket) or an inter-node mechanism (e.g. TCP/IP socket running over a network connection). When the allocation of PUs is changed to better optimize the application, the underlying support for local/remote transparency allows the application to continue to function without the need to change its application code.

Dynamic relocation could be beneficial where a communications wire or network is simply bogged down and we need to communicate in a different way. Or there may be a circumstance when shared memory resources become tight and it's more important to spend time over a communication wire than to use shared memory. Or furthermore we determine that the heap size of a given job is starting to cause problems and therefore splitting out the work and relying upon IPC is the correct choice of action. To facilitate the dynamic relocation, there are metrics for each possible communications mechanism used by the processing units. These simple metrics are used to track how much a given resource is being used and how much more taxing adding more work can be to the given situation for that resource.

FIG. **1** shows a block diagram that represents a massively parallel computer system **100** such as the Blue Gene/L computer system. The Blue Gene/L system is a scalable system in which the maximum number of compute nodes is 65,536. Each node **110** has an application specific integrated circuit (ASIC) **112**, also called a Blue Gene/L compute chip **112**. The compute chip incorporates two processors or central processor units (CPUs) and is mounted on a node daughter card **114**. The node also typically has 512 megabytes of local memory (not shown). A node board **120** accommodates 32 node daughter cards **114** each having a node **110**. Thus, each node board has 32 nodes, with 2 processors for each node, and the associated memory for each processor. A rack **130** is a housing that contains 32 node boards **120**. Each of the node boards **120** connect into a midplane printed circuit board **132** with a midplane connector **134**. The midplane **132** is inside the rack and not shown in FIG. **1**. The full Blue Gene/L computer system would be housed in 64 racks **130** or cabinets with 32 node boards **120** in each. The full system would then have 65,536 nodes and 131,072 CPUs (64 racks×32 node boards×32 nodes×2 CPUs).

The Blue Gene/L computer system structure can be described as a compute node core with an I/O node surface, where communication to 1024 compute nodes **110** is handled by each I/O node **170** that has an I/O processor connected to the service node **140**. The I/O nodes **170** have no local storage. The I/O nodes are connected to the compute nodes through the logical tree network and also have functional wide area network capabilities through a gigabit Ethernet network (See FIG. **2** below). The gigabit Ethernet network is connected to an I/O processor (or Blue Gene/L link chip) in the I/O node **170** located on a node board **120** that handles communication from the service node **160** to a number of nodes. The Blue Gene/L system has one or more I/O nodes **170** connected to the node board **120**. The I/O processors can be configured to communicate with 8, 32 or 64 nodes. The service node uses the gigabit network to control connectivity by communicating to link cards on the compute nodes. The connections to the I/O nodes are similar to the connections to the compute node except the I/O nodes are not connected to the torus network.

Again referring to FIG. **1**, the computer system **100** includes a service node **140** that handles the loading of the nodes with software and controls the operation of the whole system. The service node **140** is typically a mini computer system such as an IBM pSeries server running Linux with a control console (not shown). The service node **140** is connected to the racks **130** of compute nodes **110** with a control system network **150**. The control system network provides control, test, and bring-up infrastructure for the Blue Gene/L system. The control system network **150** includes various network interfaces that provide the necessary communication for the massively parallel computer system. The network interfaces are described further below. In the Blue Gene/L system there may also be a number of front end nodes that are similar to the service node **140**. As used herein, the term service node includes these other front end nodes.

The service node **140** communicates through the control system network **150** dedicated to system management. The control system network **150** includes a private 100-Mb/s Ethernet connected to an Ido chip **180** located on a node board **120** that handles communication from the service node **160** to a number of nodes. This network is sometime referred to as the JTAG network since it communicates using the JTAG protocol. All control, test, and bring-up of the compute nodes **110** on the node board **120** is governed through the JTAG port communicating with the service node.

The service node includes a job optimizer **142** that allocates parts of applications called jobs to execute on one or more of the compute nodes. As illustrated in FIG. **1**, the job optimizer is software executing on the service node **140**. Alternatively, the job optimizer **142** may also reside on a front end node or on another node of the system. The job optimizer may be stored in data storage **138** which may comprise a hard disk for temporary storage or a compact disk for distribution or sale. In conjunction with the job optimizer **142**, the service node **140** also has a collector **144** that includes application specific metrics **145** and hardware system metrics **146** used by the job optimizer to determine how to allocate jobs. The application specific metrics **145** described herein include various metrics that are determined by application metric plug-ins. The application specific metrics **145** and the hardware system metrics **146** are described below.

FIG. **2** illustrates a block diagram of an exemplary compute node as introduced above. FIG. **2** also represents a block diagram for an I/O node, which has the same overall structure as the compute node. A notable difference between the compute node and the I/O nodes is that the Ethernet adapter **226** is connected to the control system on the I/O node but is not used in the compute node. The compute node **110** of FIG. **2** includes a plurality of computer processors **210**, each with an arithmetic logic unit (ALU) **211** and a memory management unit (MMU) **212**. The processors **210** are connected to random access memory ('RAM') **214** through a high-speed memory bus **215**. Also connected to the high-speed memory bus **214** is a bus adapter **217**. The bus adapter **217** connects to an extension bus **218** that connects to other components of the compute node.

Stored in RAM **214** is a an application program **224**, and an operating system kernel **225**. The application program is loaded on the node by the control system to perform a user designated task. The application program typically runs in parallel with application programs running on adjacent nodes. The application **224** may be divided into one or more job(s) **226** which may be further divided into one or more processing units **228**. The application further includes one or more application metric plug-ins **229** that determines application specific metrics **145** as described further herein. The operating system kernel **225** is a module of computer program instructions and routines for an application program's access to other resources of the compute node. The quantity and complexity of tasks to be performed by an operating system on a compute node in a massively parallel computer are typically smaller and less complex than those of an operating system on a typical stand alone computer. The operating system may therefore be quite lightweight by comparison with operating systems of general purpose computers, a pared down version as it were, or an operating system developed specifically for operations on a particular massively parallel computer. Operating systems that may usefully be improved, simplified, for use in a compute node include UNIX, Linux, Microsoft XP, AIX, IBM's i5/OS, and others as will occur to those of skill in the art.

The compute node **110** of FIG. **2** includes several communications adapters **226**, **228**, **230**, **232** for implementing data communications with other nodes of a massively parallel computer. Such data communications may be carried out serially through RS-232 connections, through external buses such as USB, through data communications networks such as IP networks, and in other ways as will occur to those of skill in the art. Communications adapters implement the hardware level of data communications through which one computer sends data communications to another computer, directly or through a network.

The data communications adapters in the example of FIG. **2** include a Gigabit Ethernet adapter **226** that couples example I/O node **110** for data communications to a Gigabit Ethernet **234**. In Blue Gene, this communication link is only used on I/O nodes and is not connected on the compute nodes. Gigabit Ethernet is a network transmission standard, defined in the IEEE 802.3 standard, that provides a data rate of 1 billion bits per second (one gigabit). Gigabit Ethernet is a variant of Ethernet that operates over multimode fiber optic cable, single mode fiber optic cable, or unshielded twisted pair.

The data communications adapters in the example of FIG. **2** include a JTAG Slave circuit **228** that couples the compute

node **110** for data communications to a JTAG Master circuit over a JTAG network **236**. JTAG is the usual name used for the IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports used for testing printed circuit boards using boundary scan. JTAG boundary scans through JTAG Slave **236** may efficiently configure processor registers and memory in compute node **110**.

The data communications adapters in the example of FIG. **2** include a Point To Point Network Adapter **230** that couples the compute node **110** for data communications to a network **238**. In Blue Gene, the Point To Point Network is typically configured as a three-dimensional torus or mesh. Point To Point Adapter **230** provides data communications in six directions on three communications axes, x, y, and z, through six bidirectional links **238**: +x, −x, +y, −y, +z, and −z. The torus network logically connects the compute nodes in a lattice like structure that allows each compute node **110** to communicate with its closest 6 neighbors.

The data communications adapters in the example of FIG. **2** include a collective network or tree network adapter **232** that couples the compute node **110** for data communications to a network **240** configured as a binary tree. This network is also sometimes referred to as the collective network. Collective network adapter **232** provides data communications through three bidirectional links: two links to children nodes and one link to a parent node (not shown). The collective network adapter **232** of each node has additional hardware to support operations on the collective network.

Again referring to FIG. **2**, the collective network **240** extends over the compute nodes of the entire Blue Gene machine, allowing data to be sent from any node to all others (broadcast), or a subset of nodes. Each node typically has three links, with one or two links to a child node and a third connected to a parent node. Arithmetic and logical hardware is built into the collective network to support integer reduction operations including min, max, sum, bitwise logical OR, bitwise logical AND, and bitwise logical XOR. The collective network is also used for global broadcast of data, rather than transmitting it around in rings on the torus network. For one-to-all communications, this is a tremendous improvement from a software point of view over the nearest-neighbor **3**D torus network.

FIG. **3** shows a block diagram that illustrates the interaction of the software elements shown to reside in the service node **140** of FIG. **1** and the compute node **110** in FIG. **2**. The application **224** includes one or more application metric plug-ins **229**. The plug-ins may be a procedure or routine called by the application or the collector to determine application specific metrics **145**. The collector **144** in the service node **140** collects application specific metrics **145** from the application **224**. The application specific metrics **145** are any suitable metrics that can be determined by application plug-ins **229** associated with the application **224**. The collector passes these application specific metrics **145** to the job optimizer **142**. The application specific metrics are examined to determine if they are within the desired range of values. This may be accomplished by passing the values for the metrics to another plug-in or function written by the application developer or a configuration file could be used to define the desired range of values for the metric. The execution of the application metric plug-in may be done on a time interval set up in the application. A comparison of the metrics and desired values is then done by the job optimizer. The job optimizer may then determine how to best allocate processing units on the multi-nodal computer system.

The collector **144** collects metrics that are used by the job optimizer to dynamically allocate jobs or parts of a jobs (processing units) on a multi-nodal, parallel computer system. Application specific metric are measurable objectives or expected results of the application. Examples of application specific metrics **145** include the following: production rates, data communication rates, data quality, data availability, data throughput, result values or quality, etc. For an example of data throughput, if a distributed application is supposed to assess the quality of 10,000 manufactured widgets per hour, a metric would be defined at the application level that measures this rate. If this metric is not met, the job optimizer would analyze how to allocate the jobs to get better performance as described below. For an example of result quality, if a distributed application is trying to identify unique individuals in a video data stream through facial identification, a metric could be defined to expect the application to be able to identify at least 10 individuals per second. If this metric is not met, then the job optimizer would attempt to allocate the processing units of the job for better performance. For an example of data availability, if a distributed application is supposed to ingest data from 1000 sensors per second, and it's not ingesting at that rate, the job optimizer similarly attempts to allocate the processing units for better performance.

The collector **144** may also collect hardware system metrics (**146** FIG. **1**) in conjunction with the application specific metrics for the job optimizer to dynamically allocate jobs or parts of a jobs (processing units) on a multi-nodal, parallel computer system. The operating system kernel (**225** FIG. **2**) or other system components may assist in gathering the hardware system metrics. In contrast to the application specific metrics, the hardware system metrics are metrics related more to the hardware and are gathered by the system. Examples of hardware system metrics include the following:

1) System Metrics:
Aggregate CPU utilization across the multi-nodal system
Aggregate Memory utilization across the multi-nodal system
Aggregate network load across the multi-nodal system
Node-to-node network utilization
2) Node Metrics:
CPU utilization for a node
Memory utilization for a node
Heap size for a node
3) Application hardware metrics:
Aggregate CPU utilization by an application
Aggregate memory utilization by an application
Result throughput for the application
Result latency for the application
4) Job metrics
Aggregate CPU utilization for the job
Aggregate memory utilization for the job
Data throughput utilization for the job
Data latency for the job
5) Processing Unit (PU) metrics
CPU utilization of the PU
Memory utilization of the PU
Data throughput of the PU
Data latency for the PU

FIG. **4** is a block diagram representing a portion of the computer system **100** shown in FIG. **1**. Each node **110**A-**110**F has a job **226** containing one or more processing units **228**. The job **226** on the nodes **110**A-**110**F may collectively make up a single application or the jobs **226** may be portions of different applications. This diagram represents the interaction of data communication between processing units in the system. The lines **410** between the processing units **228** represent data communication or data sharing between the processing units. Processing units **228** within the same job **226** on the same node may also be communicating but no line is shown.

FIG. **5** is a block diagram representing two nodes, NodeA **110**A and NodeB **110**B, of a computer system similar to computer system **100** shown in FIG. **1**. FIG. **5** in conjunction with FIGS. **6** and **7** illustrate examples of dynamically allocating jobs as described and claimed herein. Running on NodeA **110**A is Job**1** **226**A that is composed of four processing units (PU**1** **228**A, PU**2** **228**B, PU**3** **228**C and PU**4** **228**D). Running on NodeB **110**B is Job**2** **226**B that is composed of two processing units (PU**5** **228**E and PU**6** **228**F). For this example, Job**1** and Job **2** combined comprise an application **224**. PU**1** **228**A and PU**2** **228**B process data from one or more inputs sources (not shown). PU**3** takes data from PU**1** and PU**2** and reduces and/or summarizes the data. PU**4** **228**D takes data from PU**3** and performs some complex statistical analysis using the data. PU**4** then publishes its results to Job**2** **228**B running on NodeB **110**B.

An example of dynamically changing the distribution of processing units will now be described with reference to FIG. **5** and FIG. **6**. In this example, the application specific metric is a data communication rate. The application metric plug-in **229** executes to determine an application specific metric **145**. In this example, we assume the application specific metric is a data rate of records sent from Job**1** **226**A to Job**2** **226**B that requires sending 1000 data records per second. The application metric plug-in determines that recently Job**1** **226**A is not processing records fast enough to send to Job**2** **226**B to meet the required data records per second. The Job Optimizer **142** (FIG. **1**) then examines the hardware system metrics **146** (FIG. **1**) to determine what resources are contributing to the problem identified by the application specific metric. In this example, the job optimizer may find a hardware system metric, such as a processor utilization metric, that indicates Job**1** is executing slowly because NodeA **110***a* is overutilized. The job optimizer then examines Job**1** and Job**2** for possible changes in allocation to optimize system resources and interprocess communication. The Job Optimizer **142** then determines to split Job**1** into two jobs to increase the output of Job**1**. The two jobs from Job**1** are shown as Job**1***a* **610** and Job**1***b* **612** in FIG. **6**. The communications between PU**3** and PU**4** is an interprocess communication that is currently local, but can also be handled over a communication link. This change in communication is preferably handled by the operating system in a way that is invisible to the processing unit as discussed above. PU**4** **228**D is moved into Job**1***b* **612** and placed on NodeC **110**C as shown in FIG. **6**. Job**1***a* **610** will now consist of PU**1**, PU**2** and PU**3** running on NodeA **110**A. Job**1***b* will consist of PU**4** running on NodeC **110**C. The result is that more CPU resources are available to PU**4** and it should be able to provide results faster to Job**2** **226**B.

FIG. **7** is a block diagram to illustrate an example of dynamically combining a job as described and claimed herein. In this example, the application specific metric is a production rate. The initial scenario for this example is as shown and described above with reference to FIG. **5**. In this example we assume that the application analyzes images taken of some widgets as they came off an assembly line. The application uses PU**1** **228**A through PU**4** **228**D running in Job**1** **226**A on NodeA **110**A, and PU**5** **228**E and PU**6** **228**F running in Job**2** **226**B on NodeB **110**B as shown in FIG. **5**. The processing units in Job**1** **226**A preprocess the images and the processing units in Job**2** **226**B perform the analysis.

The pre-processed images flow from Job**1** to Job**2** over a network connection (because the jobs are running on different nodes). If some or all of the image files start to grow in size (because of higher resolution image capture devices or adjustments made to the captures devices), the network connection between these two nodes might become overloaded. The application metric plug-in **229** collects the metric data that allows the job optimizer to determine the application was not analyzing widgets images at the desired rate. This metric may be determined, for example, by examining the results of Job**2**, i.e. how many widgets have been analyzed per second. The Job Optimizer **142** (FIG. **1**) examines the application specific metrics **147** from the collector (FIG. **1**) and examines Job**1** and Job**2** for possible changes in allocation to optimize system resources and interprocess communication. Once the application problem is identified by the application specific metric, the Job Optimizer then uses the hardware system metrics above to determine what hardware resources are affecting the application specific metric and how to overcome the problem. For example, the network between Job**1** and Job**2** may be found to be overloaded by a node-to-node network utilization metric. Alternatively, it may be found that sending the data over the network requires to much overhead to keep up with the desired data rate by looking at a processor utilization metric. The Job Optimizer **142** then determines to combine Job**1** and Job**2** into a single job to alleviate the overloaded network to increase the production rate. The combined job is shown in FIG. **7** as Job**3** **226**C.

A hardware system metric (**146** in FIG. **1**) can be used in combination with one or more application specific metrics **145** to identify a job or PU that is not executing within a desired range. In our example above, the application specific metric of widgets per second may be compared with memory utilization of the node which is a hardware system metric. The two could be combined by dividing memory utilization by widgets per second, where memory utilization is expressed as a percentage. Then job optimizer may determine the application is using too much memory per widget when the ratio exceeds some desired combination metric such as 5%. For example a 90% memory utilization and a widget rate of 55 per sec would result in an acceptable value (a value of 1.64% which is less than 5%) while a 60% memory utilization and a widget rate of 8 per sec would indicate a problem (a value of 7.5% which is greater than 5%). The Job Optimizer would then determine how to dynamically change the allocation of the processing units to overcome the problem identified by the combination metric. In addition, after the identifying the problem with the combined metric, the Job Optimizer may optionally also examine other hardware system metrics to identify resource problems affecting the combined metric to determine how to dynamically change the allocation of the processing units on the compute nodes based on the collected metrics as described herein.

The previous examples described splitting a job running on one node into two jobs that run on two nodes and combining PUs into a job that runs on a single node. Similarly, a job running on one node can be split into two jobs that run on one node where there may be a performance benefit to do this in some cases. For example, on nodes with multiple processors, breaking things up into multiple jobs may allow for better exploitation of the multiple processors. This would be done in a manner similar to that described above.

FIG. **8** shows a method **800** for dynamically adjusting allocation of processing units on a multi-nodal computer system according to embodiments herein. The steps in method **800** are preferably performed by the collector and job optimizer executing on the service node and/or the compute nodes of the system. First the job optimizer starts execution of the application with one or more application metric plug-ins on one or more compute nodes of the system (step **810**). The application may comprise one or more jobs, where each job may comprise one or more processing units. The collector then collects appropriate application specific metrics from the processing units of the application (step **820**). The job optimizer then analyzes the collected metrics (step **830**). Next, the job optimizer checks the collected metric to determine if they are in the accepted range (step **840**). If the metrics are in the accepted range (step **840**=yes) the return to step **810**. If the metrics are not in the accepted range (step **840**=no) then identify the jobs and/or processing units affecting the performance as indicated by the application specific metric out of range (step **850**). Then examine the hardware system metrics to identify a what resource problems are contributing to the application specific metric identified above (step **855**), and access the potential job and processing unit reallocations that could be used to dynamically change the allocation of the processing units on the compute nodes based on the collected metrics and identified resources to optimize system resources and interprocess communications (step **860**). Determine whether to combine processing units or split processing units to alleviate problem associated with the collected application metric and the identified resource (step **870**). If it is determined to combine the jobs (step **870**=combine) then combine one or more processing units into a single job (step **880**) and return to step **810**. If it is determined to split the jobs (step **870**=split) then split a job into multiple processing units on separate nodes to alleviate poor performance of the job (step **890**) and return to step **810**. The method is then complete.

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a “circuit,” “module” or “system.” Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device. A computer readable signal medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable signal medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device. Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). Aspects of the present invention are described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks. The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

As described above, embodiments provide a method and apparatus dynamically changes the allocation of processing units on a multi-nodal computer system. One skilled in the art will appreciate that many variations are possible within the scope of the claims. Thus, while the disclosure has been particularly shown and described above, it will be understood by those skilled in the art that these and other changes in form and details may be made therein without departing from the spirit and scope of the claims.

The invention claimed is:

1. An apparatus comprising:

a plurality of nodes of a multi-nodal computer system, wherein the plurality of nodes are connected by a plurality of networks, where each of the plurality of nodes has at least one central processing unit (CPU) coupled to a memory;

an application having a plurality of jobs, each with at least one processing unit executing on the plurality of nodes;

an application metric plug-in associated with the application executing on the plurality of nodes that determines an application specific metric that is a measurable objective of the application in order to determine how to best allocate the jobs on the system;

a collector that collects the application specific metric;

a job optimizer that dynamically changes the allocation of processing units on the plurality of nodes based on the collected application specific metric to optimize system resources and interprocess communications; and

wherein the application specific metric is a measurable objective of the application comprising data throughput wherein the application specific metric further comprises a measurable objective of the application chosen from the following:

production rates, data communication rates, data quality, data availability, a result value and a result quality.

2. The apparatus of claim **1** wherein the job optimizer dynamically changes the allocation of the processing units by combining at least two processing units from jobs on different nodes into a job on a single node of the plurality of nodes.

3. The apparatus of claim **1** wherein the job optimizer dynamically changes the allocation of the processing units by splitting a job into multiple jobs on different nodes of the plurality of nodes.

4. The apparatus of claim **1** wherein the job optimizer dynamically changes the allocation of the processing units by splitting a job into multiple jobs on a same node to utilize multiple processors of the same node.

5. The apparatus of claim **1** wherein the application metric plug-in is initiated by the collector.

6. The apparatus of claim **1** wherein the application metric plug-in is initiated by the application.

7. The apparatus of claim **1** further comprising a hardware system metric that is used in conjunction with the application specific metric to determine how to best allocate the jobs on the system.

8. An apparatus comprising:

a plurality of nodes of a multi-nodal computer system, wherein the plurality of nodes are connected by a plurality of networks, where each of the plurality of nodes has at least one central processing unit (CPU) coupled to a memory;

an application having a plurality of jobs, each with at least one processing unit executing on the plurality of nodes;

an application metric plug-in associated with the application executing on the plurality of nodes that determines an application specific metric that is a measurable objective of the application in order to determine how to best allocate the jobs on the system, wherein the measurable objective comprises: production rates, data communication rates, data quality, data availability, data throughput, a result value and a result quality;

a collector that collects the application specific metric from the application metric plug-in; and

a job optimizer that analyzes the application specific metric, and when the application specific metric is outside a desired range, the job optimizer performs the steps of:

identifying jobs affecting the application specific metric;

examining hardware system metrics to identify resources affecting the application specific metric;

assessing potential job and processing unit relocations in order to determine how to best allocate the jobs on the system; and

dynamically changing the allocation of the processing units on the compute nodes based on the collected metrics and identified resources to optimize system resources and interprocess communications by combining at least two processing units executing on different nodes into a job on a single node.

9. An article of manufacture comprising software stored on a non-transitory computer-readable storage medium comprising:

an application having a plurality of jobs, each with at least one processing unit for execution on a node of a multimodal computer system;

an application metric plug-in associated with the application executing on the plurality of nodes that determines an application specific metric that is a measurable objective of the application in order to determine how to best allocate the jobs on the system;

a collector that collects the application specific metrics and examines a hardware system metric to identify resource problems affecting the application specific metric;

a job optimizer that dynamically changes the allocation of processing units on the plurality of nodes based on the collected application specific metric and the hardware system metric to optimize system resources and interprocess communications; and

wherein the application specific metric is a measurable objective of the application comprising data throughput wherein the application specific metric further comprises a measurable objective of the application chosen from the following: production rates, data communication rates, data quality, data availability, a result value and a result quality.

10. The article of manufacture of claim **9** wherein the job optimizer dynamically changes the allocation of the processing units by combining at least two processing units from jobs on different nodes into a job on a single node of the plurality of nodes.

11. The article of manufacture of claim **9** wherein the job optimizer dynamically changes the allocation of the processing units by splitting a job into multiple jobs on different nodes of the plurality of nodes.

* * * * *